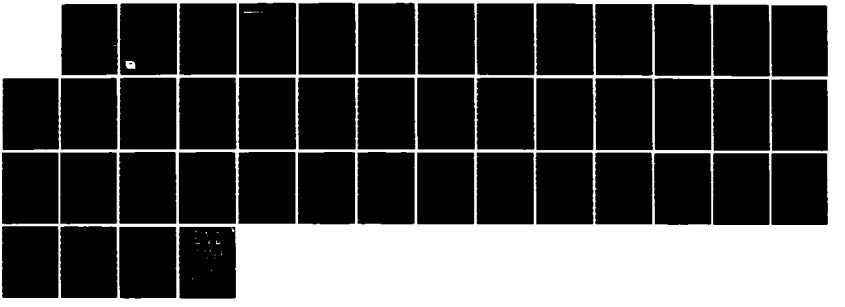
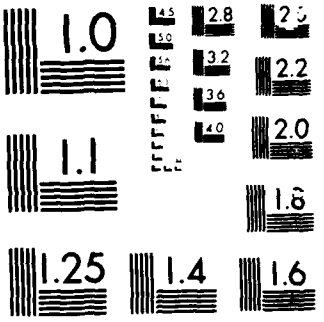
AD-A166 664 THE STATUS OF VERIFICATION TECHNOLOGY FOR THE ADA (TRADEMARK) LANGUAGE..(U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA K A NYBERG ET AL. JUL 85 1/1

UNCLASSIFIED IDA-P-1859 IDA/HQ-85-30214 MDA903-84-C-0031 F/G 9/2 NL





IDA PAPER P-1859

THE STATUS OF VERIFICATION TECHNOLOGY FOR THE Ada* LANGUAGE

Karl A. Nyberg
Audrey A. Hook
Jack F. Kramer

July 1985

Prepared for
Office of the Under Secretary of Defense for Research and Engineering

INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, Virginia 22311

IDA Log No. HQ 85-30214

The work reported in this document was conducted under contract MDA 903 84 C 0031 for the Department of Defense. The publication of this IDA Paper does not indicate endorsement by the Department of Defense, nor should the contents be construed as reflecting the official position of that agency.

This Paper has been reviewed by IDA to assure that it meets high standards of thoroughness, objectivity, and sound analytical methodology and that the conclusions stem from the methodology.

SECURITY CLASSIFICATION OF THIS PAGE AD-A166 664

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY [redacted]	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) IDA Paper P-1859	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Institute for Defense Analyses 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION OUSDRE (DoD IDA Management Office)
6c. ADDRESS (City, State, and ZIP Code) 1801 N. Beauregard Street Alexandria, VA 22311	7b. ADDRESS (City, State, and ZIP Code) 1801 N. Beauregard Street Alexandria, VA 22311
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Office of the Under Secretary of Defense Research & Engineering 8b. OFFICE SYMBOL (If applicable) OUSDRE	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER MDA 903 84C 0031
8c. ADDRESS (City, State, and ZIP Code) Ada Joint Program Office 1211 Fern St, Room C-107 Arlington, VA 22202	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. T-5-306 / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

The Status of Verification Technology for the Ada Language

12. PERSONAL AUTHOR(S)
Karl A. Nyberg, Audrey A. Hook, Jack F. Kramer

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM ____ TO ____	July 1985	38

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Ada programming language, verification, software development methodologies, verification technology.

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This report provides a detailed research and development plan for the development and deployment of an Ada verification capability. The background, history, and goals of both Ada and verification are discussed. Specific recommendations for integrating verification technology in the software development process (particularly with Ada) are presented. The necessary research items to be pursued in support of the goals are also presented.

This report consists of two major sections - the text and supporting appendices and references. The appendices generally describe efforts being undertaken in various areas and are annotated with status and points of contact for further information.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

IDA PAPER P-1859

THE STATUS OF VERIFICATION TECHNOLOGY FOR THE Ada LANGUAGE

Karl A. Nyberg
Audrey A. Hook
Jack F. Kramer

July 1985

INSTITUTE FOR DEFENSE ANALYSES

Contract MDA 903 84 C 0031
Task T-5-306

PREFACE

This paper discusses the subject of verification of Ada programs and the requirements for verification technology in the context of secure systems. It examines the areas where research has either started or is needed in order to develop a verification capability for Ada programs. A phased plan for developing verification technology has been proposed; estimates of time and level of effort are provided. Additionally, Appendix A identifies the Ada development projects that will require verification of software; Appendix B describes some of the methodology development efforts; Appendix C identifies many of the current research efforts in verification technology while Appendix D identifies those efforts that focus on the Ada language. Appendix E provides a list of verification Working Groups and the contact for each group who can provide additional information. This research was conducted for IDA by Karl A. Nyberg of Verdix Corporation.

Many of the concepts discussed in this report were also discussed during two Verification Workshops conducted by IDA; however, the plan of action presented in this report was not formally included in the deliberations of the workshop participants. The variety of research efforts and working groups that are currently addressing the problems of verification technology and applying verification to Ada programs is indicative of the level of interest demonstrated by academia, industry and government.

1. Introduction

This report provides a detailed research and development plan for the development and deployment of an Ada verification capability. The background, history, and goals of both Ada and verification are discussed. Specific recommendations for integrating verification technology in the software development process (particularly with Ada) are presented. The necessary research items to be pursued in support of the goals are also presented.

This report consists of two major sections — the text and supporting appendices and references. The appendices generally describe efforts being undertaken in various areas and are annotated with status and points of contact for further information.

2. Background

The need for application of verification technology to the development of software written in Ada is becoming critical. The Ada mandate for software development is taking hold, and other issues relevant to the Department of Defense (particularly security) must be examined in the light of this mandate. The development and subsequent publication of the Trusted Computer Security Evaluation Criteria (TCSEC) by the Department of Defense Computer Security Center (DoDCSC) on the heels of that of the Ada Language Reference Manual (ANSI/MIL-STD-1815A-1983) [Ada] has reinforced the need to address the issue of Ada verification. A recent report provided by the DoDCSC [Rowe] describes some of the interests and goals of the Center in this area.

As an example of the urgency for integration of verification and software development in Ada, there are a number of efforts within the DoD that have requirements for a high degree of security (obtained through the use of verification techniques) and for implementation in Ada. (Some of these efforts are listed in Appendix A.) Failure to integrate these two technologies could jeopardize the goals toward which these efforts have been directed.

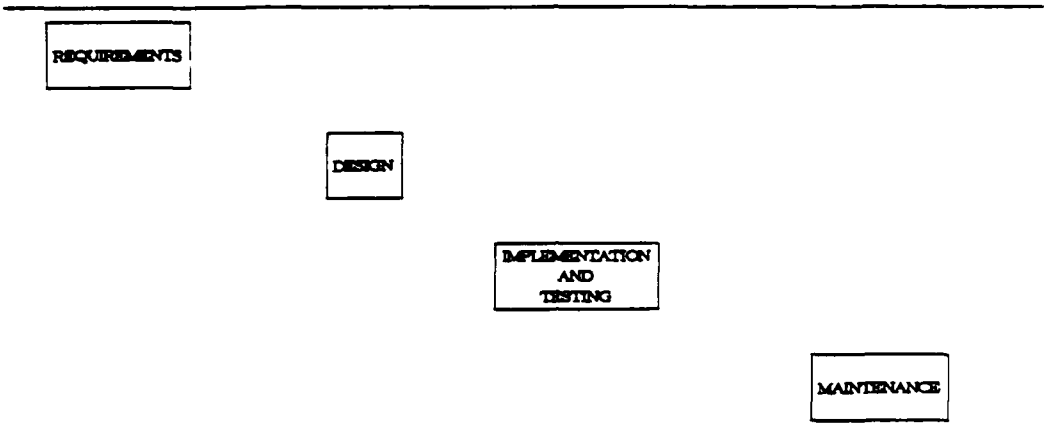
2.1. Programming Languages

Programming languages have evolved from the low level machine codes and assembly languages of the 1950s to the more abstract "high level" languages of the 1980s (Pascal, C, Ada, etc.). Along the way, each new language has left its particular mark upon the computing milieu. The presumption about evolution is that each succeeding generation of languages is better (in some objective sense of the word) than the previous generation. Some of the features that have been used to distinguish new programming languages have been in the areas of data abstraction, conditional constructs, concurrency, suitability for a particular type of application (business, scientific, list processing, communications or operating systems).

Although the evolution of programming languages has provided many benefits for the software development profession, there are also drawbacks to the evolution of programming languages. Foremost among them, and perhaps the *raison d'etre* for the development of the Ada language, is the incredible effort required over the life cycle of software development projects to support the multitude of languages and hardware systems. Another major drawback is the limited scope of hardware upon which programs written in a particular programming language may be executed. Aside from such languages as C, FORTRAN, and COBOL few programming languages (and virtually no systems programming languages) are widely available on a multitude of hardware configurations. The goal of a standardized Ada is that programs written to execute on one set of hardware will execute on other sets of hardware with a minimum of modification.

2.2. Software Development Methodologies

The software development lifecycle can be broken down into four general phases: requirements, design, implementation and testing, and maintenance. A common mechanism for graphically displaying these phases is the standard waterfall diagram shown below, and basic aspects of each phase are presented. As this lifecycle is well understood and applied, each area will not be further elaborated here, but reference is made to Section 6, where a method for the integration of verification techniques into the software development lifecycle is presented.



Aside from the generally accepted practices of software development such as top-down development (also called structured design, hierarchical development, etc.) a particular emphasis is being placed on using what is known as an object oriented approach in the development of Ada software. [Berard] This approach applies particularly during the design phase of software development and attempts to place more emphasis on the development and use of abstract data types in the software development cycle.

2.3. Verification Technology

Verification is the process in which mathematical reasoning is used to show that a system satisfies its requirements. The two particular types of verification that are distinguished are those of design verification and of implementation verification. Design verification is the process of showing that the design of a system (expressed as a model of the system in a specification language) satisfies the requirements of the system. The requirements of the system are usually expressed as constraints or invariants on the system's behavior — relations between variables making up the state of the system. Design verification is performed in a non-procedural language that expresses *what* is being accomplished, not *how* it is being accomplished. Design verification is no guarantee that the resulting system will indeed have the properties, only that the model of the system (as expressed in the design) does. Implementation verification is required to bridge the gap between the design and the executable program. Implementation verification is used to show in a constructive manner that an implementation satisfies its requirements by actually developing a system. The requirements of the system are expressed as pre- and post-conditions [Floyd67], and the post-conditions are proven from the pre-conditions and the symbolic execution of the constructed system. These pre- and post-conditions are obtained from the design, and any attempts to apply implementation verification techniques in the absence of a design (or worse yet, in a *post hoc* fashion, after the program has been developed) are subject to frustration and futility. These techniques must be integrated into the software development lifecycle (see Section 6 for such a discussion), and not applied independently of such lifecycle.

Verification technology has progressed to the point where it is now possible to apply the techniques to the development of small systems for both design and implementation. However, use of the technology is limited by the languages available for describing such systems, which were developed as many as ten years ago, and do not reflect the state of the art in programming language technology. Another difficulty preventing the use and application of verification technologies is the cost involved. Use of the tools requires individuals with a high level of formal education, and requires a major commitment to computing resources in order to be able to be applied effectively. For this reason, larger projects desiring to apply verification have stopped at the design verification level.

In the past, applied verification technology has been driven more by the security community than any other community. (For a listing of automated verification tools and their applications, see Appendix C.) There are also some related issues in the area of reliability that indicate that the sorts of language limitations necessary for allowing the proof of properties relevant to reliability are the same sorts of limitations as those required for performing verification.

3. Integrating Ada Verification in the Software Development Process

In the application of verification technology to Ada, there are three approaches that can be taken:

- Follow Current Practices
- Design Verification with Ada
- Integrated Design and Implementation Verification with Ada

Issues surrounding each of these alternatives are discussed in the following sections.

3.1. Current Practices

Ignoring the particular requirement of Ada with respect to formal verification is the simplest of the alternatives available. It requires no development of additional tools, no training in additional† technologies, no modification of the current development approach. From the point of view of secure applications, the implementation language used in the development of an A1 system (currently the highest rating given by DoDCSC) is irrelevant. Only design verification must be performed (using one of the "approved" verification systems — Gypsy, HDM, or Affirm), and only informal correspondence with the implementation must be given using a method such as verification†† and validation (V&V). This approach does make the informal correspondence significantly harder to do than if a consistent set of languages were used, but is currently being used in the development of such A1 secure systems as the Honeywell SCOMP, so is not infeasible.

The drawback to this alternative is that it is necessary for the system developers to be fluent in multiple methodologies, systems, and languages, all differing from one another if the use of some form of design verification is desirable. While this may be an acceptable solution in the very short term, it will not be sufficient for the long term, due to the complexities of the various systems. Furthermore, this fragmentation of the development teams along the lines of specialization (design, verification, implementation) will increase the overhead associated with program development.

---

† The key word in this sentence is "additional" — the tools and technologies for verification will still have to be available, and individuals still trained in their use and application.

†† Unfortunate, but the same word is used in differring contexts by different people to mean different things. Use of "verification" in this context is what has lead to the concept of *post hoc* formal verification, which is totally inappropriate.

3.2. Design Verification with Ada

A second alternative that can be taken is the use of Ada as a design language, both for implementing standard requirements and design methodologies, as well as in preparation for design verification. The requirements and design phases could all be done in Ada (or an Ada Program Design Language [PDL]), and with the development of supporting verification tools, design verification performed. This would significantly decrease the gap between the design being verified and the implementation. It would also only require the developers to be familiar with a much smaller number of methodologies, permitting a more unified approach to the development of verifiably secure systems than the current disjointed approach being applied.

The glaring drawback to this alternative is the lack of automated tools for the support of design and for verification, as well as the lack of a suitable language definition for design specification. Although some tools have been developed for the use of Ada as a design language, many of them are little more than subsets of Ada front ends. There are no tools available for the verification of designs written using an Ada PDL, nor do there appear to be any in development. The only existing serious specification language particularly directed toward Ada (Anna [Anna]) has only limited application at the current time. Anna does not contain a sufficient set of constructs for performing design verification, but rather is geared more towards the development of runtime checking. Furthermore, Anna is not integrated with an Ada compiler, but must be run as a separate tool. An additional drawback to this approach is the lack of a formal semantic definition of the constructs in the portion of the language used for design (the non-procedural portions) verification. Without such a semantic definition it remains impossible to reason about program designs and prove properties of these designs.

3.3. Integrated Design and Implementation Verification with Ada

The final alternative to applying verification technology to Ada would be to perform implementation verification as well as design verification. This alternative would provide the *most* integrated approach of verification in the software development process, directly following design verification as one step in the total verification process. It allows the greatest amount of formalism to be applied in the development of software, and allows verification to be applied down to the source code level.† However, this alternative is currently infeasible from both a theoretical and a practical point of view (particularly so for Ada). It inherits all the difficulties of performing design verification mentioned above, as well as some Ada unique problems. In addition to a semantics for the non-procedural portions of the language, the procedural portions must also have their semantics formally defined. Additional tools for processing the programs, their specifications, and symbolic execution of the programs must be developed. Tools for proving the correspondence between the specification and the implementation (verification conditions resulting from the symbolic execution) must also be developed.

4. Goals

This section describes goals to be pursued in developing and deploying an Ada verification capability. Each goal is described in terms of those aspects of it that may be completed in the near-term (up to three years, now to 1988) and long-term (more than three years, past 1988). This time span is chosen as the definition of the Ada language is currently "frozen" until 1988, and it would be beneficial to have some recommendations available to the language maintenance organization when the time for review of the language arrives. The first two alternatives given in the previous section (current practices, and design verification with Ada) help achieve short term goals, while the third alternative (integrated design and implementation verification with Ada) is a more long term goal.

† Verification could conceivably be applied to even lower levels, to microprocessor instruction sets, and even to gate levels, although the techniques and mechanisms required for this level of verification (to say nothing of the hardware resources required)

4.1. Formal Semantics of Ada

A major goal that must be obtained before any significant application of verification technology may be made with Ada is the development of a formal semantics of the language. As a formal semantics of the language was not developed in concert with the development of the language, it might not be possible to develop a semantics for the entire language. However, if semantics for only a portion of the language can be developed, the implication is that only that portion of the language may be used for systems that will require verification. There are a number of efforts already underway, however they suffer from disjoint interests, approaches, and are not well coordinated with one another in order to provide a single definitive result. (Some of these efforts are noted in Appendix E, Sections 12.1.3 and 12.3.) Once the semantics are developed, progress may be made toward development of automated tools for the support of verification.

4.2. Specification Language for Ada

Another goal that must be attained before the verification of Ada programs may be undertaken is the development of a specification language for Ada. Both a design specification language and an implementation specification language (hopefully correlated in order to allow both forms of verification to be applied) will have to be developed. Progress has been made in this area with the Anna language, (see Appendix D, Section 11.1) and effort is underway in more general areas of specification (see Appendix E, Section 12.1.4).

4.3. Automated Tools for Verification

A goal in the area of verification (as well as software development) has been the development of supporting automated tools. These tools aid the developers by performing as much of the detailed bookkeeping and other automatable work as possible, allowing the developers to use their time in other, more creative and productive means. Another distinct advantage that results from the development of automated tools (used as a primary argument for the development of prototypes) is the opportunity to use the tools as an embodiment of the issue at hand to provide for further understanding of the situation. While applying verification to Ada, the development of such tools as verification condition generators and symbolic evaluators would allow testing of the semantics being developed to insure that they can be applied in such an environment, and that they indeed correspond to the execution of the constructs in the language.

There are two possible mechanisms by which automated tools for the support of verification can be developed — modification of existing tools, and the development *ab initio* of a verification system. The former is clearly a short-term goal, while the latter one of much longer term.

4.3.1. Modification of Existing Verification Tools

The modification of existing verification tools provides a mechanism whereby the investment in those tools can be leveraged to make as quick as possible a return on that investment in the area of Ada verification. Many of the tools that have been developed (expression simplifiers, symbolic execution and interpretation, theorem provers) are in part or in whole language-independent, and should be amenable to minor modification for application in the Ada verification arena. Another benefit to use of modified tools would be the ability of the tools to serve as a prototype testbed for evaluation of proposed semantics of the language, and allow experimentation without the large investment required by the full-scale development of a new system.

---

are currently beyond the state of the art.

For current efforts addressing this goal see Appendix D, particularly Sections 11.3 and 11.4, and Appendix E, Section 12.1.2.

4.3.2. Development of New Verification Tools

The development of new verification tools provides the opportunity to make a cohesive set of tools particularly geared toward the needs of systems developers using Ada. It also provides an opportunity to develop the tools themselves in Ada (leveraging off of the experience of previous developments), as well as to possibly integrate the tools into environments particularly targeted for Ada development (e.g., ALS). Experience with Ada is somewhat limited at the moment, and development systems for Ada are still at a quite immature stage. This would indicate that caution should be taken before attempting to commit significant resources to such a development. Another reason for hesitancy in committing to such an effort is the lack of an established formal semantics for Ada, a necessary prerequisite to such effort.

5. Research

This section lists particular areas/sub-areas of Ada and verification for which additional research may be necessary, both separately and in conjunction with one another.

5.1. Verifiability of Ada

Perhaps the greatest obstacle to the application of verification technology to Ada is the fact that the language was not designed as a verifiable language. Furthermore, the size and complexity of the language make it difficult to understand, from the point of view of verifiability, not only the constructs in the language but also interactions between many of the constructs. One solution that has been proposed for dealing with this complexity is the enforcement of a "verifiable subset" of the language in situations where verification is applied to Ada software development.

The greatest need in research in Ada for verification is the development of a formal semantics for the language. Without such a semantics, it is impossible to do symbolic evaluation of programs and provide a verification condition generator. Several efforts have been undertaken in the development of semantics for portions of the language, [Luckham80, Pneuli82, Barringer82, Gerth82, Gerth83], but additional work needs to be performed to coordinate these efforts and unify their results.

In addition to the semantics of the language, the concurrent aspects of the language need further investigation, since they cause considerable difficulty in use of the language. Lastly, the indeterminancy features of the language need to be resolved in such a way as to make the language "predictable".

5.1.1. Language Subsets

Although the issue of language subsets for compilers has long been considered *anathema* to the Ada Joint Program Office, such a requirement is not applicable in the area of verification. It is wholly appropriate to use that portion of the language that is amenable to verification, and use any validated compiler for the generation of executable code. Determination of appropriate subset(s) to be used in software development requireing verification could use only that "predictable" subset of the language or follow the "clusters" approach (see Section 12.1.6).

5.1.2. Formal Semantics

The Ada language divides quite nicely into two areas in which its semantics can be developed — the sequential and the concurrent aspects of the language. Further subareas of research in these two areas are described in the following sections.

5.1.2.1. Sequential Constructs in Ada

For the most part, the sequential constructs in Ada are similar to those found in other high level languages (Pascal, C, CLU, Gypsy), and their semantics can be easily determined from the semantics of the corresponding constructs in the other languages. There are two particular areas that will require additional effort: generics and exceptions.

5.1.2.1.1. Generics

Aside from the issue of concurrency in the language, the second most difficult portion of the language is that of generics [Ernst, Young80]. While the use of generics is a very powerful concept, care must be taken to ensure that the semantics are well-defined, and that they are amenable to automatic processing for use in verification tools.

5.1.2.1.2. Exceptions

Exceptions, *per se*, are not a new feature in programming languages [Good78, Goodenough75]. In Ada however, exceptions have been promoted to a more prominent place in the language, and must be treated with as much care as other constructs in the language, not as an add-on. Although some work has been done in developing a semantics for Ada expressions [Luckham80], this work needs to be integrated with the development of the semantics for the remainder of the language.

5.1.2.2. Concurrent Constructs in Ada

As with exceptions, a semantics for the concurrent constructs in Ada has been under investigation [Barringer82]. And, like the work in the semantics of exceptions, this work needs to be integrated with the efforts in semantics of the remainder of the language.

5.1.3. Model of Concurrency

As mentioned above, part of the difficulty in obtaining an appropriate semantics of the concurrent constructs of the Ada language has been due to the choice of the model. The Ada model for concurrency is that of the rendezvous. A model more commonly used in languages for verification is that of message passing. The two are computationally equivalent, however, the message model becomes more difficult for verification unless restrictions on other communications are present (e.g., no global variables, *a la* Gypsy), and this restriction was deemed too stiff for the language definition. Additional work needs to be done to see if the rendezvous model can be made more amenable to verification, and/or if a message passing mechanism could be used in Ada.

5.1.4. Predictability

Aside from the non-determinism resulting from the concurrency in the language, another characteristic of the Ada language is its non-predictability. This characteristic arises from such statements in the Language Reference Manual as in Section 11.6 (2) *Exceptions and Optimization:*

> When, on the other hand, the order of certain actions is not defined by the language, any order can be used by the implementation.

or Section 6.4 (6) *Subprogram Calls*

> The parameter associations of a subprogram call are evaluated in some order that is not defined by the language. Similarly, the language rules do not define in which order the values of in out or out are copied back into the corresponding actual parameters (when this is done).

There are a number of solutions to the difficulty encountered in predictability of programs. The most straightforward approach would be to eliminate these difficulties from the language. If the language definition doesn't specify an ordering in such instances, it should be appropriate to pick a particular ordering (one that would be amenable to verification), and recommend that it be instantiated as the accepted one.† The adoption of a firm policy that any Ada-implemented system to be certified at the B2 or higher level must be implemented in "predictable Ada" might be considering.

5.2. Verification

In the area of verification, there are a number of issues that relate to the continuing language developments that have not yet found solutions. In particular, the areas of concurrency, fault tolerance, and real number arithmetic remain to have significant strides made before becoming viable for use in development of systems.

5.2.1. Concurrency

Verification of general concurrency is a difficult task, due to the usual "flat" address space available in most programming languages. With suitable restrictions on access to variables and communications between concurrent processes (such as in Gypsy), some verification of concurrent processes can be performed. It would be beneficial to investigate other models of concurrency, such as the rendezvous mechanism of Ada, to find ways of making them more amenable to verification.

5.2.2. Fault Tolerance / Reliability

Fault tolerance has long been a well understood and accepted practice in hardware design, but the techniques thus used have not been applied in the area of software, or even systems design. One difficulty in this area is the lack of a suitable model of computation for fault tolerance in which verification could be investigated.

---

† It is worth noting that it is not only the verification community that has expressed interest in solution to this issue — compiler vendors have the same desire to see the language definition tightened.

5.2.3. Floating Point Arithmetic

Floating point arithmetic has been an area in verification that has been almost totally ignored. The major reason for this situation is the fact that an appropriate idealization of floating point arithmetic has not been developed that would be appropriate for a number of hardware implementations and amenable to analysis for verification efforts.

5.3. Ada and Verification

Aside from the other issues that arise in the application of verification technology to the development of software in Ada, a serious effort should be placed on looking at the effect of the language on verification methodologies. Although it appears at first glance that verification should be easily integrated into the software development process (see Section 6 below), there may remain peculiarities with regard to Ada, due to the advanced features present in the language. These features, not present in other languages designed for verification, might inhibit the choice of methodologies available for verification.

6. Verification and the Software Development Process

This section describes an approach for making verification an integral part of the software development process.† It recommends where those issues raised in Section 3 can be addressed, and suggests efforts to be undertaken as well as policy to be encouraged in order to achieve the desired goals, both in the short and the long terms. The various areas in which verification is applied directly follow the waterfall diagram provided in Section 2.2.

6.1. Requirements

Although there is limited application of verification technology to the requirements phase of a software development effort, it is appropriate that verification be taken into consideration at this stage. If verification is intended for an effort, it will be beneficial to start with a formal statement of the requirements. Such a statement could serve as a "contract" between the developers of a system and any contracting party. An example of an informal requirements presentation (the KAIS FEU) in [Smith84] is turned into a formal one in [Good84a].

6.2. Design

In the design phase, verification can be applied to show that the design indeed satisfies the requirements developed in the previous step. This step can be taken in conjunction with the design development, although many times the design is developed without any regard for verification, and the verification is not considered until after the design is completed. It is, however, at this stage that the pressure to produce a running system increases, and projects often split into two directions — prototype development and verification. This split has the unfortunate result of not allowing any feedback from the design verification to affect the design, and subsequently the implementation. As a result, the system whose design was verified may not correspond to the system implemented. This appears to be the case reported in [SIFT]. A design (and implementation) verification where the design verification indeed corresponds to the implementation is presented in [Smith81], while the design verification of the KAIS FEU mentioned above is presented in [Good84e].

† A particularly good treatise on the methodology for applying verification technology in the software development process particularly oriented to the TCSEC A1 level of certification can be found in [Good84d].

6.3. Implementation and Testing

With an approach to program development that doesn't allow (or encourage) the implementation to progress any faster than the implementation verification, application of verification technology in the implementation and testing phase is straightforward. As each succeeding level of software is developed, it is proven correct with respect to its specification. By the time the development reaches the lowest levels of the implementation, each of the higher levels will have been verified in turn.

One area where verification technology that has been overlooked in the software development process is the automated development of test cases, particularly for modules. Implementation verification requires the development of verification conditions, obtained by a symbolic evaluation of the program. This symbolic evaluation identifies the various paths possible during a program's execution, and could be used to determine a set of test inputs that could exhaustively exercise particular modules.

Another application of verification technology is in the area of "reusable" software. Once a particular set of software (e.g., packages, libraries, etc.) were developed and verified, they could be used in the software development process without requiring the reimplementation and reverification of their functionality. If a particular application can leverage off the work in a similar application, perhaps the benefit of reusable theories [Good82c] and verified libraries can also be put to use.

6.4. Maintenance

Maintenance in the software development lifecycle is what occurs after the software has been "released", and recommendations for changes, updates, fixes, etc. start being encountered. It is this phase that lends justification to the claim of the non-sequential aspect of the waterfall diagram presented in Section 2.2. Although at some point in the system development process it may be desirable to take the current system, designate it as a prototype, and proceed from scratch with a new development, more often the current system is modified to add an additional feature or modify the manner in which a particular feature operates. Most often, this modification is performed by feeding back to the implementation and testing phase. However, to keep the entire development lifecycle in mind, it is appropriate to return to the requirements phase, and indicate how the change affects the requirements, the design, *and* the implementation and testing phase. This is an area that is also ripe for application of verification technology.

The most applicable principle of verification technology for use in the maintenance phase of software development is that of incremental methods [Moriconi77]. Similar in concept to the principle of separate compilation (available in many languages and compilers, and made explicit in Ada), the principle of incremental methods for verification technology states that only those portions of a system that have been changed need to be re-verified for the system to remain "proven". As (for example) the reimplementation of (the body of) a function or procedure in a package in Ada does not require the recompilation of those other units that depend only upon its interface specification, so the reimplementation of a unit that continues to meet its pre- and post-condition specifications does not require the reverification of any other units.

7. Research and Development Plan

This section describes the particular research and development efforts that are recommended for the deployment of Ada verification capabilities. Time schedules, estimated levels of effort, and interdependencies are provided both in a narrative and a tabular form. Such estimates must be tempered with the understanding that independent efforts such as would be expected of such large projects would incur additional startup overhead and initial familiarity learning curves. It must also be understood that a coordination of these efforts will also require additional effort. Finally, these estimates are only that —

estimates.† Although based upon the experiences of others in performing similar work in the verification community, these are extrapolations into an unknown arena, with many possible pitfalls present.

The schedule as presented is predicated on a number of assumptions. Foremost among those is the ability to start the efforts immediately. While this may be a fallacious assumption, it is possible that the existing efforts may be leveraged off of in order to overcome the initial startup and learning overhead. Secondly, the schedule attempts to complete as much work by the end of 1988 in order to be able to provide input to the language maintenance committee, which will be considering language changes in that time frame.

7.1. Formal Semantics of Ada

The development of the Ada language itself was a multiyear task, requiring the resources of numerous individuals. As a result, the developments of a semantics for the language ought to have the same flavor, and is likely to have a similar extent. Time should be allowed for review of any results, and for experimentation with prototype verification systems to investigate the appropriateness of the developed semantics.

The two areas of Ada for which semantics must be developed are the sequential and the concurrent parts of the language. The split is a natural one, and lends itself to a useful separation of concerns.

7.1.1. Sequential Portion

A number of efforts have begun to address this issue (see Appendix E, Sections 12.1.3 and 12.3), although no results have been published or reviewed as of yet. It is estimated that this effort will require ten staff-years over two and one half calendar years, including serious outside review.

7.1.2. Concurrent Aspects

Once a semantics for the sequential portion of the language has begun, work on the concurrent aspects should be started and integrated with the sequential work. It is a vital necessity that these two effort be done in concert, so as not to develop a semantics for the sequential portion of the language that is incompatible with a semantics for the concurrent aspects. This effort is estimated at six staff-years over two calendar years.

7.2. Specification Language

Although not specifically designed as a language for the proof of programs concerning Ada, the Anna specification language provides an excellent starting point for the development of a specification language for verification work. Modifications in order to more adequately support verification oriented constructs should be investigated, and the language extended to handle these constructs. Furthermore, integration of a specification mechanism to describe concurrency should be performed.

† These estimates should probably be considered as the midpoint in a range, running from possibly twenty percent lower to twenty percent higher.

7.2.1. Sequential Portion

With the Anna language as a basis for the development of a specification language, it will be necessary to extend the language to include more proof-theoretic constructs necessary for doing implementation verification. This effort is estimated at three staff-years over one calendar year.

7.2.2. Concurrent Aspects

In concert with the development of the formal semantics for the concurrent aspects of the language, and after the work on the specification language for the sequential portion of the language has begun, work on integrating specification constructs for the concurrent aspects of the language should be integrated into the specification language. As with the work on the semantics, this effort should be done in concert with the effort on the sequential portion of the specification language to insure their compatibility. This effort is estimated to require two staff-years over one calendar year.

7.3. Automated Verification Tools

There are two possibilities for the development of automated tools in the support of verification activities — prototype tools for the short-term, and (possibly ALS) integrated tools for the long term.

7.3.1. Prototype Tools

A number of efforts are underway in the development of prototype tools for the support of verification with Ada. Some known efforts are listed in Appendix D, particularly Sections 11.3 and 11.4. These sorts of efforts should be encouraged in the short term (up to two-three years calendar time) in order to gain more experience with the similarities and differences between Ada and other languages, and the way in which verification impacts the use of Ada as a development language. The support of two such projects over the next two and one half years could amount to some ten staff-years.

7.3.2. Integrated Tools

Plans should be laid for the long term development of an Ada verification capability that would be integrated with the remainder of a development system currently envisioned for software development in Ada. A significant amount of understanding and experience with applied verification using Ada would have to be obtained, and used in the development of such a system.

Current verification systems are the result of many people's efforts over a significant period of time, and it ought not to be expected that attacking the Ada problem will be any simpler. A critical mass of approximately six individuals working over a three year span is probably a lower bound on the estimate necessary.

7.4. Tables

The following tables provide the information just presented in a graphic tabular form. The first table describes the recommended time schedules for the various tasks, and the second table the staffing level. Again, these tables are predicated on starting immediately, and attempting to complete as much effort as possible to allow input to the language maintenance committee by 1988.

Calendar	85	85	86	86	86	86	87	87	87	87	88	88	88	88	89	89	89	89
Quarter	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Semantics																		
Sequential	•	—	—	—	—	—	—	—	—	†								
Concurrent			•	—	—	—	—	—	—	†								
Spec. Language																		
Sequential	•	—	—	†														
Concurrent			•	—	—	†												
Tools																		
Prototype	•	—	—	—	—	—	—	—	—	†								
Integrated							•	—	—	—	—	—	—	—	—	—	—	†

Calendar	85	85	86	86	86	86	87	87	87	87	88	88	88	88	89	89	89	89
Quarter	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Semantics																		
Sequential	4	4	4	4	4	4	4	4	4	4								
Concurrent			3	3	3	3	3	3	3	3								
Spec. Language																		
Sequential	3	3	3	3														
Concurrent			2	2	2	2												
Tools																		
Prototype	4	4	4	4	4	4	4	4	4	4								
Integrated							6	6	6	6	6	6	6	6	6	6	6	6

8. Appendix A - Current Ada Efforts Requiring Verification

This appendix lists the known efforts that require the use of Ada and have some requirements for verification of one sort or another. Current status of the efforts and a relevant point of contact are identified.

8.1. Inter-Services/Agency Automated Message Processing Environment (I-S/A AMPE)

The AMPE project requires the use of Ada as a PDL for the system design as well as the accreditation of the system as an A1 system according to the TCSEC.

8.2. Department of Defense Intelligence Information System (DoDIIS)

The DoDIIS is currently investigating the use of Ada on a project-wide basis for re-implementation of it database systems. The are no known verification requirements per se at this time.

Defense Intelligence Agency
Attn. RSE1B (Cpt. Frank Stellar)
Washington, DC 20301-6111
(202)-373-3012

8.3. World-Wide Military Command and Control System Information System (WIS)

The WIS has a mandated requirement for a majority of the system to be written in Ada, and an unstated requirement for multi-level security, but no explicit requirement for verification.

Col. William Whitaker
WIS JPMO/ADT
Washington, DC 20330-6600
(703)-285-5065

8.4. Army Secure Operating System (ASOS)

The Army Secure Operating System (formerly Military Computer Family Operating System - McFOS) is a program with the goal of developing an A1 operating system coded in Ada with the express purpose of running Army tactical applications also coded in Ada. The system is currently undergoing prototyping with the top level specification being done inhouse in HDM Special, and the preliminary software compiled using the ICSC compiler. Plans are to migrate to the Ada Language System (ALS) as it becomes available.

Eric Anderson
TRW
R2/1124
1 Space Park
Redondo Beach, CA 90278
(213)-535-5776

8.5. Secure Communications Protocol

The Network Division of the Department of Defense Computer Security Center is investigating the issue of the development of secure communications protocols using Ada and using verification techniques to prove the security of the protocols.

Ken Rowe
DoDCSC, C3
9800 Savage Road
Ft. George G. Meade, MD 20755-6000
(301)-859-6790
DDN: rowe@tycho

9. Appendix B - Verification in the Software Development Process

This appendix lists the know efforts attempting to determine appropriate methodologies / metaphors for the use of verification in the software development process. It lists those that are specific to the Ada language as well as those that are not.

9.1. ACM National SIGAda Design Methodology Committee

This committee chair is coordinating the Ada Verification work with other efforts underway in the area of the methodology of Ada software development. A presentation of the results of the first IDA workshop will be presented at the SIGAda conference this summer.

Alton Brintzenhoff
SYSCON Corp.
3990 Sherman Street
San Diego, CA 92110
(619)-296-0085
DDN: sci-ada@eclb

9.2. Verification in Life Cycles

See the corresponding entry in Appendix D, Section 11.4.5, for a working subgroup resulting from the First IDA Workshop on Formal Specification and Verification of Ada.

9.3. Ada-Europe Formal Methods Working Group

The Formal Methods Working Group of Ada-Europe is preparing an Ada style guide for the use of a verifiable subset of Ada to be used in the development of programs intended to be verified. (It appears that this is the same group that is working on a formal semantics for the language - see Section 12.3.)

10. Appendix C - Current Verification Systems

This appendix lists current efforts in verification, particularly those that are unrelated to the verification of Ada. (For a list of those efforts that are particularly related to Ada, see Appendix D.) A good starting point for comparative evaluations can be found in [Cheheyl81]. Another evaluation is currently underway with a report due out this fall.† The current status and point of contacts for each of the efforts are listed with each effort.

In addition, some of the more visible applications of each of these systems are listed, with the current status and/or results of the applications. Points of contact are also provided where available.

10.1. Gypsy Verification Environment (GVE)

The following are the first three paragraphs of "Using the Gypsy Methodology" [Good84b]. They adequately describe the Gypsy methodology.

> The Gypsy methodology is an integrated system of methods, languages, and tools for designing and building formally verified software systems. The methods provide for the specification and coding of systems that can be rigorously verified by logical deduction to always run according to specification. These specification, programming, and verification methods dictated the design of the program description language Gypsy. Gypsy consists of two intersecting components: a formal specification language and a verifiable, high level programming language. These component languages can be used separately or collectively. The most important characteristic of Gypsy, however, is that it is fully verifiable. The entire Gypsy language is designed so that there exist rigorous, deductive proof methods for proving the consistency of specifications and programs. The methodology makes use of the *Gypsy Verification Environment* (GVE) to provide automated support. The GVE is a large interactive system that maintains a Gypsy program description library and provides a highly integrated set of tools for implementing the specification, programming, and verification methods.
>
> The Gypsy methodology may also be applied strictly to the design phase of system development. For example in certain applications, particularly in the security domain, it is considered desirable to prove that a system's specifications possess specific properties. In Gypsy these properties would typically be stated as lemmas or as the specifications on an abstract data type, and the verification of the design would consist of demonstrating that the high level specifications satisfy these lemmas and type specifications. For more on proving properties of specifications see [Good84c].
>
> The effective range of application of the methodology depends on the applicability of Gypsy to a particular problem. Gypsy is suitable for a wide range of general and systems programming applications. Gypsy was derived from Pascal and retains much of the wide applicability of Pascal. One major exception is the absence of floating point in Gypsy. Gypsy, however, does have major facilities for exception handling, data abstraction, and concurrency that are not present in Pascal. During its development, the methodology has been used successively in several substantial experimental applications. These include message switching systems, selected components of an air traffic control system, communication protocols, security kernels, and monitoring of inter-process communication.

For further information on the Gypsy language and the Gypsy Verification Environment, contact:

> Dr. Donald Good
> Institute for Computing Science and Computer Applications
> 2100 Main
> The University of Texas

† The effort is entitled "Verification Assessment". The point of contact is Richard A. Kemmerer, Computer Science Department University of California, Santa Barbara, CA 93106. (805)-961-4232. DDN: dick@ucla-locus.

Austin, TX 78712
(512)-471-1901
good@utexas-20

Some of the more notable applications of Gypsy and the GVE are listed in the following sections. A listing of Gypsy applications can be found in [Good83].

10.1.1. Encrypted Packet Interface

The Encrypted Packet Interface (EPI) [Good82a] was the first major trial application of the GVE. The EPI is a device that sits between a host and an IMP on the ARPANET. Properties of proper encryption and decryption of TCP (4.0) packets between the two sides of the EPI were specified and proved. The code implemented interoperated successfully across the ARPANET with a companion interface developed and implemented independently in conventional assembly code at BBN.

10.1.2. Message Flow Modulator

The Message Flow Modulator (MFM) [Good82b] is an LSI-11 based security filter that monitors message flow from the Ocean Surveillance Information System (OSIS). The MFM monitors all messages for occurrences of certain security sensitive phrases. It has been demonstrated successfully in an operational environment.

10.1.3. KAIS Front End Unit

The KAIS (Korean Air Intelligence System) Front End Unit (FEU) is a security filter that releases messages segments from a high security STREAMLINER system to a lower security Combat Support System (CSS). The FEU is currently under development at the University of Texas and an attempt to meet all relevant certification requirements under the A1 criteria of the DoDCSC. Some reports on the KAIS FEU include the requirements [Smith84], design (security model and top level specifications) [Good 84a], and design proofs [Good84e].

10.2. Affirm

What follows is the introduction of a brief paper put together for the Verification Workshop that took place in February. It documents the current status of the AFFIRM system.

> The AFFIRM Program Verification System originated at the University of Southern California Information Sciences Institute (ISI). It is an experimental system for the algebraic specification and verification of abstract data types and Pascal-like programs using these types. AFFIRM-85 is an enhanced version of AFFIRM that is being developed at General Electric Corporate Research and Development Center (GE-CRD). This paper briefly describes two major extensions that will be completed early in 1985. The primary purpose of these and several minor extensions is to enable the use of AFFIRM in carrying out a larger part of the software development process than previously has been possible.
>
> The Hierarchical Support mechanism of AFFIRM-85, has been designed to support the approach to hierarchical organization of abstract data types and their implementations, supplemented by a method of connecting equational implementations to Pascal procedures and functions. The Hierarchical Support mechanism, has been implemented.
>
> The heart of the AFFIRM system is a natural deduction theorem prover for the interactive proof of data

type properties and verification conditions. A Reusable Theorems mechanism that allows the user to keep track of what assumptions a theorem depends on, even across session boundaries has been designed and is currently under implementation.

David R. Musser
General Electric Research & Development Center
KW C265A
P.O. Box 8
Schenectady, NY 12301
(518)-387-5964
DDN: musser@ge-crd

10.3. Hierarchical Development Methodology (HDM)

HDM, the Special specification language, and the tools supporting the methodology and language are currently undergoing enhancements and revisions. From [Melliar-Smith85], goals of the new system include:

- Specifications in first order predicate calculus with second order capability
- Strong type checking with overloading
- Parameterized modules with semantic constraints
- User interface based on multi-window screen editor [EMACS]
- Theorem proving by reduction to propositional calculus, with decision procedures for common theories
- Hoare sentences and code proof
- Multilevel security (MLS) checking by information flow analysis

HDM and Special have been used in the development of a number of projects, some of which are listed below. In addition, the Army Secure Operating System being developed by TRW (See Appendix A, Section 8.5) will be using HDM.

P. Michael Melliar-Smith
Computer Science Laboratory
SRI International
333 Ravenswood Ave.
Menlo Park, CA 94025
melliar-smith@sri-csl

10.3.1. System Implemented for Fault Tolerance (SIFT)

Although a significant amount of work was completed under the auspices of this effort [Levitt83b], it is not clear that the effort indeed provided the sorts of results that were being sought or in the particular areas being investigated. In a peer review of the results obtained in the effort [SIFT], a significant amount of disagreement over the claimed results surfaced.

10.3.2. Kernelized Secure Operating System (KSOS)

KSOS was begun with the goal of designing, implementing, and proving a secure operating system. [Berson]. Some twenty levels of abstraction were written in Special describing the design of the system. An implementation was then developed that runs insufficiently fast to be of any use for development of significant applications.

10.3.3. Probably Secure Operating System (PSOS)

PSOS was designed using HDM by organizing a collection of approximately 20 hierarchically related modules [Feiertag]. The basing mechanism underlying the PSOS was that of a capability machine. No implementation has been constructed.

10.3.4. Honeywell Secure Communications Processor (SCOMP)

Honeywell, in concert with the DoDCSC, developed the SCOMP, a communications processor derived from the Honeywell Level 6 computer, with additional hardware for memory management and protection. The SCOMP was specified in Special (with trusted processes in Gypsy), verified, then implemented in C & Pascal. It has received an A1 rating from DoDCSC.

10.4. Formal Development Methodology (FDM)

No information has been received on FDM.

Cooper@MIT-Multics

10.5. Modula Verification System

The Modula Verification System (MVS) is an attempt to apply verification techniques and principles to Modula. In this application, there has been no serious attempt to keep to the definition of the Modula language, and as a result, programs written in "verifiable Modula" are not processible by Modula compilers. One major feature of Modula that was omitted was that of concurrency. In addition to the parser a verification condition generator and theorem prover have been implemented.

(The effort in the MVS has also resulted in an attempt to convert the MVS into an Ada verification system. This latter effort is documented in Appendix D, Section 11.5.)

Professor Ray Hookway
Department of Computer Engineering and Science
Case Western Reserve University
Cleveland, OH 44106
(216)-368-2800
DDN: hookway%case@csnet-relay

10.6. Verus Verification System

The VERUS Verification System, developed at Compion (formerly DTI) is now undergoing product review

by Gould in preparation for marketing. It had been used at Compion for several internal developments.

Liza Nowell
Product Planning
Gould Computer Systems
6901 W. Sunrise Blvd.
Fort Lauderdale, FL 33313
(305)-797-5733

11. Appendix D - Current Ada Verification Efforts

This appendix lists the current known efforts in verification that are particularly aimed toward the Ada programming language and toward developing systems for automated support of verification. It also includes those efforts that are more directed toward the language itself, but looking at verification issues as well. Status of the various efforts, as well as points of contact are provided.

11.1. Anna Specification Language

From the Preface to the Preliminary Reference Manual for Anna [Luckham84]:

> Anna is a language extension of Ada to include facilities for formally specifying the intended behavior of Ada programs. It is designed to meet a perceived need to augment Ada with precise machine-processable annotations so that well established formal methods of specification and documentation can be applied to Ada programs.

A number of documents in addition to the reference manual are in process, including: (1) an introduction to the use of Anna; (2) transformations from annotations to Ada runtime checks; and (3) an axiomatic semantics of Anna.

Current (and planned) tools include: (1) syntax analyzers, structured editors, tools for detecting simple kinds of errors; (2) a runtime checking system that will translate most annotations into Ada runtime checks. A formal verification system is regarded as a longer-term undertaking.

Professor David Luckham
Stanford University
ERL 428 CSL
Stanford, CA 94305-2192
(415)-497-1242
DDN: luckham@su-ai

11.2. Asphodel Specification and Design Language

An effort in Europe has lead to the development of a specification language, Asphodel [Hill], similar in style to Anna for the use of design and specification of Ada programs. The language is designed to be more in the style of the Vienna Development Method [Bjorner] rather than that of the algebraic style [Guttag].

Alec Hill
Central Electric Generating Board
Computing and Information Systems Department, Laud House,
20 Newgate Street, London EC1A 7AX, U.K.

11.3. Cornell Attribute Grammar Tools

The Cornell Attribute grammar tools have been considered for use in the development of an Ada verification system. This development could occur by generating a formal description of Ada annotated with attributes, and run through the various tools. This idea was first suggested in [Reps84], and is currently under investigation.

Ryan Stansifer
Cornell University
Ithaca, NY 14850

11.4. Prototype Ada Verification System

Verdix Corporation is currently investigating the development of a prototype Ada verification system based upon the modification of its own validated Verdix Ada Development System (VADS)[tm] and the Gypsy Verification Environment developed at The University of Texas. The front end of the VADS is being modified to accept not only Ada, but Anna-like annotations as well. Once the syntactic and semantic parsing have been completed, the resulting Diana structures will be passed through a "verification filter" to check that an acceptable "verifiable subset" of the Ada language is being used. This will then result in a database of structures suitable for reading into the GVE for verification condition generation and theorem proving. This effort is expained in greater detail in [McHugh85].

Karl Nyberg
Verdix Corporation
7655 Old Springhouse Rd.
McLean, VA 22102
(703)-448-1980
DDN: nyberg@eclb
UUCP: vrdxhq!karl

11.5. Modula Verification System Conversion

The Modula Verification System developed at Case Western is currently undergoing modification to attempt to turn it into an Ada verification system. The basic premise is that a lot of verifiable Modula will be similar to verifiable Ada. A particular example cited is that of the similarity private types in Ada and exported types in Modula. The programs accepted by the verification system are in a modified Ada syntax to include pre- and post-conditions and other constructs necessary for verification, and the system will output a version of the programs acceptable as input to an Ada compiler. No semantic checking of types, overloading, or any other constructs that a validated compiler is expected to catch are made.

The current system consists of a parser (based upon Herm Fischer's grammar), the front end of a verification condition generator, and supports library constructs. The semantic parts of the verification condition generator are currently under design and development.

Professor Ray Hookway
Department of Computer Engineering and Science
Case Western Reserve University
Cleveland, OH 44106
(216)-368-2800
DDN: hookway%case@csnet-relay

11.6. NSA Internal Efforts

A number of efforts are going on within the Department of Defense Computer Security Center. Those that are listed here deal only with the particulars of the development of verification technology for Ada.

Verdix, VADS are trademarks of Verdix Corporation.

(Other NSA efforts are listed in Appendix A, Section 8.7.) The two particular areas in Ada verification that are being pursued are the investigation of the verifiability of Ada — how restricted would a verifiable subset have to be, and the development of tools for performing the automated verification of Ada. Efforts underway in the second area include both inhouse and pending proposed efforts.

Brian Holland
DoD Computer Security Center, C3
9800 Savage Road
Ft. George G. Meade, MD 20755-6000
(301) 859-6968
DDN: brian@tycho

12. Appendix E - Working Groups

This appendix lists current known efforts aimed at investigating the development, deployment and usage verification techniques in the development of systems using Ada.

12.1. IDA Workshops on Formal Specification and Verification of Ada

The Institute for Defense Analysis (IDA) is conducting workshops on formal specification and verification of Ada. The first workshop was held in March 1985, and the second is planned for mid to late summer 1985. The first Workshop on Formal Specification and Verification of Ada resulted in a set of proceedings (currently in draft form) and the establishment of several working subgroups. These subgroups, and the status of their efforts are presented in the following sections. Although some of the subgroups might be more appropriately listed in other Appendices, they are included here for consistency, and cross referenced where appropriate.

Dr. Jack Kramer
Institute for Defense Analyses
1801 N. Beauregard St.
Alexandria, VA 22311
(703) 845-2263
DDN: kramer@eclb

12.1.1. Secure Systems

The goal of this working subgroup is to attempt to determine the suitability of Ada for use in the development of secure systems, particularly those intended to be submitted to the DoDCSC for ratings at or above the B2 level. The chair of this subgroup is attempting to coordinate efforts with the Ada Task Force (see Appendix A, Section 8.7.1) which has a similar charter.

Margie Zuk
MITRE Corporation
Burlington Road
Bedford, MA 01730
(617)-271-7590
DDN: security!mmz@mitre-bedford

12.1.2. Near Term Verification

The goal of this working group is to investigate the near term needs and solutions for dealing with Ada verification. In particular, automated tools for the support of verification in the near term are being investigated. Participation in the group through dissemination of network mail has begun.

In addition to the efforts uncovered during the IDA workshop, several verification and Ada related projects are being investigated. These include:

- Ada based fault-tree analysis being done by Nancy Leveson at UCI.
- An Ada to Special flow tool being developed by Compusec.
- Application of IBM's verification methodologies known as the "Clean Room" approach to Ada

It is hoped that the working group will meet prior to the next IDA meeting, possibly in early July at RTI.

John McHugh
Research Triangle Institute
Box 12194
Research Triangle Park, NC 27709
(919)-541-7327
DDN: mchugh@utexas-20

12.1.3. Formal Semantics

This subgroup is involved in investigating the areas of a formal semantics for the language, both sequential and concurrent. Contacts with other organizations, particularly in Europe, who have similar interests have been made, and plans are to convene a one-day meeting this summer, hopefully just before the second IDA workshop.

Norm Cohen
SofTech, Inc.
705 Masons Mill Business Park
1800 Byberry Road
Huntingdon Valley, PA 19006
(215)-947-8880
DDN: ncohen@eclb

12.1.4. Specification Languages

No response has been received from this chair.

Friedrich von Henke
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
(415)-859-2560
DDN: vonhenke@sri-csl

12.1.5. Verification in Life Cycles

This subgroup is looking into how to integrate formal verification techniques with other software development practices in the software life cycle. Coordination with other groups pursuing similar interests (such as the ACM group mentioned in Appendix B, Section 8.1) is intended. An interim meeting is planned before the next IDA workshop.

Ann Marmor-Squires
TRW, Defense Systems Group
2751 Prosperity Avenue
Fairfax, VA 22031
(703) 876-8170
DDN: marmor@isi

12.1.6. Clusters

This working group is investigating an Odyssey-specific approach to the development of multiple, possibly overlapping subsets of Ada for use in various software developments. From discussions with the chairman, it appears that there is little involvement from others outside of the company.

Ryan Stansifer
Odyssey Research Associates, Inc.
408 E. State Street
Ithaca, NY 14850
(607) 277-2020
DDN: rplatek@eclb

12.2. Ada Task Force

The DoDCSC has convened an Ada Task Force to investigate the suitability of Ada in the development of secure computing systems. Little is known about the efforts of this group, as the head of the task force (listed below) has denied access to any minutes or proceedings of the groups only meeting to date.

Marv Schaefer
DoDCSC, C3
9800 Savage Road
Ft. George G. Meade, MD 20755-6000
(301)-859-6380
DDN: schaefer@isi

12.3. European Efforts on Formal Semantics

An effort is underway in Europe for the development of a draft Ada language formal definition (Ada FD). This effort is supported by the EEC, and is being performed at Dansk Datamatik Center. The major goals of the project are [EEC]:

To obtain as concise a definition of the full ANSI Ada language as is today feasible, in a form which

(0) May serve as a reference for questions on Ada,

and is suitable for further research on the following topics:

(1) formal work in the areas of proof systems for Ada programs,

(2) correct development of correct Ada interpreters and compilers,

(3) the meaningful generantion and verification of Ada test programs, including validation of the ACVC test suite, and

(4) the derivation of informal, but precise, unambiguous Ada reference manuals for various user groups,

in order to help provide:

- input to the ongoing standardization work on Ada, in particular to support the ISO future review of the

Ada standard, and

- a worthy, broad, and commonly accepted candidate for the formal component of a future Ada ISO Standard.

Some subsidiary objectives include:

- To help unite various approaches to the informal, and semi-formal descriptions of Ada (by studying how to relate the proposed Ada FD to e.g. the NYU SETL interpreter for Ada).

- To further develop and research engineering methods suitable for the precise definition of large, complex software systems (by calling on a wide community of computer scientists to take part both in the actual Ada FD development, and its review), and thereby

- to further propagate the use of formal methods in software engineering.

Contracting Agency:

Rudolf W. Meijer
Commission of the European Communities
Information Technology and Telecommunications Task Force
A25 9/6A
Rue de la Loi 200
B-1049 Brussels, Belgium
tel: + 32 2 235 7769
telex: 21877 comeu b
telecopier: +32 2 235 0655
DDN: rmeijer@eclb
UUCP: ...!decvax!mcvax!hrc63!rmeijer

Contractor:

Kurt W. Hansen
Dansk Datamatik Center
Lundtoftevej 1C
DK-2800 Lyngby, Denmark
tel +45 2 872622
telex 37704 ddc dk
telecopier +45 2 872217
DDN: khansen@eclb

13. References

[Ada] - *Ada Programming Language*, ANSI/MIL-STD-1815A, Department of Defense, 22 January 1983.

[Akers83] - Akers, Robert L., *A Gypsy-to-Ada Program Compiler*, Technical Report 39, Institute for Computing Science, The University of Texas at Austin, Austin, TX 78712, December 1983.

[Barringer82] - Barringer, H., Mearns, I., "Axioms and Proof Rules for Ada Tasks", *IEEE Proceedings*, Volume 29(E), Number 2, pp. 38-48, March 1982.

[Berard] - Berard, Edward V. *An Object Oriented Design Handbook for Ada Software, EVB Software Engineering, Inc. 451 Hungerford Dr. #701, Rockville, MD* 20850, 1985.

[Berg82] - Berg, H. K., Boebert, W. E., Franta, W. R., Moher, T. G., *Formal Methods of Program Verification and Specification*, Prentice-Hall, Inc., Englewood Cliffs, NY 07632, 1982.

[Berson] - Berson, Thomas A., and Barksdale, G. L. Jr., *KSOS — Development Methodology for a Secure Operating System*, Ford Aerospace and Communicatiosn Corporation, Palo Alto, CA.

[Bjorner] - Bjorner, D. and Jones, C. B., "The Vienna Development Method : The Meta Langauge", *Lecture Notes in Computer Science*, #61, Springer-Verlag.

[Boyer79] - Boyer, Robert S., Moore, J Strother, *A Computational Logic*, Academic Press, 1979.

[Boyer80] - Boyer, Robert S., Moore, J Strother, *A Verification Condition Generator for Fortran*, Technical Report CSL-103, SRI International, June, 1980.

[Boyer84] - Boyer, Robert S., Moore, J Strother, "Proof-Checking, Theorem–Proving, and Program Verification", *Contemporary Mathematics*, Volume 29, pp. 119-132, 1984.

[Cheheyl81] - Cheheyl, M. H., Gasser, M., Huff, G. A., Millen, J. K., "Verifying Security", *ACM Computing Surveys* 13(3):279-340, September, 1981.

[EEC] - *The Draft Formal Definition of ANSI/MIL-STD 1815A Ada*, EEC Multiannual Programme, Project No. 782, Annex 1, Version 14-12-1984, Dansk Datamatik Center, Lundtoftevej 1C, DK-2800 Lyngby, Denmark.

[Elspas72] - Elspas, B., Levitt, Karl N., Waldinger, Richard J., Waksman, A., "An Assessment of Techniques for Proving Program Correctness", *ACM Computing Surveys*, 4(2):97-147, 1972.

[Ernst] - Ernst, G.W. and Hookway, R.J., *Specification and Verification of Generic Program Units in Ada*, Department of Computer Engineering and Science, Case Institute of Technology, Case Western University, Cleveland Ohio.

[Evans83] - Evans, Arthur Jr., Butler, Kenneth J., Goos, G., Wulf, Wm. A., *Diana Reference Manual, Revision 3*, Tartan Laboratories, Pittsburg, PA, February 28, 1983.

[Feiertag] - Feiertag, Richard J. and Neumann, Peter G., *The Foundations of a Provably Secure Operating System (PSOS)*, SRI International, Menlo Park, CA.

[Floyd67] - Floyd, Robert W., "Assigning Meanings to Programs", *Mathematical Aspects of Computer Science*, Proceeding of a Symposium in Applied Mathematics, American Mathematical Society 19, pp. 19-32, Providence, RI, 1967.

[Gerhart80] - Gerhart, Susan L., *Fundamental Concepts of Program Verification*, AFFIRM Memo-15-SLG, University of Southern California Information Science Institute, Marina Del Rey, CA 90291, February 18, 1980.

[Gerth82] - Gerth, R. "A Sound and Complete Hoare Axiomatization of the Ada Rendesvous", *Proceedings of the 9th International Colloquium on Automata, Languages, and Programming*, Lecture Notes in Computer Science 140, Sringer Verlag, pp. 252-264, 1982.

[Gerth83] - Gerth, R. and deRoever, W. P., "A Proof System for Concurrent Ada Programs", RUU-CS-83-2, Rijksuniversiteit Utrecht, January 1983.

[Good78] - Good, Donald I., Cohen, Richard M., Hoch, Charles G., Hunter, Lawrence W., Hare, Dwight F., *Report on the Language Gypsy, Version 2.0*, Technical Report ICSCA-CMP-10, Institute for Computing Science The University of Texas at Austin, Austin, TX 78712, September 1978.

[Good79] - Good, Donald I., Cohen, Richard M., and Keeton-Williams, James, *Principles of Proving Concurrent Programs in Gypsy*, Institute for Computing Science and Computer Applications, The University of Texas at Austin, Austin, TX 78712, January 1979.

[Good82a] - Good, Donald I., "The Proof of a Distributed System in Gypsy", in *Proceedings of the 15th IBM/University of Newcastle upon Tyne Joint Seminar: Formal Specifications*, September, 1982. (Also Technical Report # 30, Institute for Computing Science, The University of Texas at Austin, Austin, TX 78712.)

[Good82b] - Good, Donald I., Siebert, Ann E., and Smith, Larry M., *Message Flow Modulator Final Report*, Technical Report ICSCA-CMP-34, Institute for Computing Science, University of Texas at Austin, Austin, TX 78712 December, 1982.

[Good82c] - Good, Donald I., *Reusable Problem Domain Theories*, Technical Report 31, Institute for Computing Science, The University of Texas at Austin, Austin, TX 78712, September 1982.

[Good83] - Good, Donald I., *Gypsy Applications, Internal Report # 111, Institute for Computing* Science, The University of Texas at Austin, Austin, TX 78712, October 1983.

[Good84a] - Good, Donald I., Siebert, Ann E., and Smith, Lawrence M., *KAIS FEU Design - Volume I Security Model Top Level Specifications*, Internal Note # 146-A, Institute for Computing Science, The University of Texas at Austin, Austin, TX 78712, September, 1984.

[Good84b] - Good, Donald I., DiVito, Bendetto L., and Smith, Michael K., *Using The Gypsy Methodology*, Draft Documentation, Institute for Computing Science, The University of Texas at Austin, Austin, TX 78712, June, 1984.

[Good84c] - Good, Donald I., *Mechanical Proofs about Computer Programs*, Technical Report # 41, Institute for Computing Science, The University of Texas at Austin, Austin, TX 78712 March, 1984.

[Good84d] - Good, Donald I., *Structuring a System for A1 Certification*, Internal Note #145-A, Institute for Computing Science, University of Texas at Austin, September 7, 1984.

[Good84e] - Good, Donald I., *KAIS FEU Design — Volume II, Proofs*, Internal Note #147-A, Institute for Computing Science, University of Texas at Austin, September 7, 1984.

[Goodenough75] - Goodenough, John B., "Exception Handling: Issues and a Proposed Notation", *Communications of the ACM*, 18(12):683-696, December 1975.

[Gregory] - Gregory, Samuel T. and Knight, John C., *A New Linguistic Approach to Backward Error Recovery*, Department of Computer Science, University of Virginia, Charlottesville, VA 22903.

[Guttag] - Guttag, John et al., "Abstract Data Types and the Development of Data Structures, *Communications of the ACM* 20(6):396-404, June 1977.

[Hedrick83] - Hedrick, Charles, *ELISP: A Large Address Space Implementation of LISP for the DECSYSTEM-20*, Rutgers University Computer Science Department, 1983.

[Hill] - Hill, A. D., *Asphodel - An Ada Compatible Specification and Design Language*, (unpublished manuscript), Central Electricity generating Board, Computing and Information SYstems Department, Laud House, 20 Newgate Streeet, London EC1A 7AX, U.K.

[Hoare69] - Hoare, C. A. R., "An Axiomatic Basis for Computer Programming", *Communications of the ACM*, 12(10):576-581, October 1969.

[Hoare76] - Hoare, C. A. R., and Wirth, Niklaus, "An Axiomatic Definition of the Programming Language Pascal", *Acta Informatica*, :2, 1976.

[Hoare81] - Hoare, C. A. R., "The Emperor's Old Clothes", 1980 ACM Turing Award Lecture, *Communications of the ACM*, 24(2):75-83, February 1981.

[Ichbiah79] - Ichbiah, J. D., Barnes, J. G. P., Heliard, J. C., Krieg-Brueckner, B., Roubine, O., Wichmann, B. A., "Preliminary Ada Reference Manual" and "Rationale for the Design of the Ada Programming Language", *ACM SIGPLAN Notices*, 14(6), June 1979.

[IDA] - *DRAFT Proceedings of the First IDA Workshop on Formal Specification and Verification of Ada*, HQ85-29920/1, Institute for Defense Analyses, 1801 N. Beauregard St., Alexandria, VA 22311, May, 1985.

[Knight] - Knight, John C. and Grine, Virginia S., *Symbolic Execution of Concurrent Ada Programs*, Department of Computer Science, University of Virginia, Charlottesville, VA 22903.

[LNRC] - *Formal Definition of the Ada Programming Language*, Institut National de Recherche en Informatique et en Automatique, November, 1980.

[Landwehr81] - Landwehr, Carl E., "Formal Models for Computer Security", *ACM Computing Surveys*, 13(3):247-278, September, 1981.

[Levitt83a] - Levitt, Karl N. *The Need for Design Verification in Fault-Tolerant Systems*, Computer Science Laboratory, SRI International, Menlo Park, CA 94025. in Proceedings of the 1983 Mission Assurance Conference.

[Levitt83b] - Levitt, Karl N., et al., *Investigation, Development, and Evaluation of Performance Proving for Fault-Tolerant Computers*, SRI International, Menlo Park, CA 94025, August 1983.

[Luckham77] - Luckham, David C., *Program Verification and Verification-Oriented Programming* American Elsevier, New York, pp. 783-793, 1963.

[Luckham80] - Luckham, David C. and Polak, Wolfgang, "Ada Exception Handling: An Axiomatic Approach", *ACM Transactions on Programming Languages and Systems*, 2(2):225-233, April 1980.

[Luckham81] - Luckham, David C., von Henke, Friedrich W., "Program Verification at Stanford", *ACM SIGSOFT Software Engineering Notes*, 6(3):25-27, July 1981.

[Luckham84] - Luckham, David C., von Henke, Friedrich W., Krieg-Brueckner, Bernd, Owe, Olaf, *Anna - A Language for Annotating Ada Programs, Preliminary Reference Manual,* Technical Report No. 84-261, Program Analysis and Verification Group, Computer Systems Laboratory, Stanford University, Stanford, CA 94305, July 1984.

[McCarthy63] - McCarthy, John, *A Basis for a Mathematical Theory of Computation,* North-Holland, Amsterdam, pp. 33-70, 1963.

[McCarthy67] - McCarthy, John, and Painter J., "Correctness of a Compiler for Arithmetic Expressions", In Schwartz, J. T. (editor), *Proceedings of a Symposium in Applied Mathematics,* Vol 19, pp 33-41, American Mathematical Society, 1967.

[McGettrick83] - McGettrick, Andrew D., *Program Verification Using Ada,* Cambridge Computer Science Texts - 13, Cambridge University Press, 1983.

[McHugh84] - McHugh, John, *Towards Efficient Code from Verified Programs,* Technical Report ICSCA-40, Institute for Computing Science, University of Texas at Austin, March 1984.

[McHugh85] - McHugh, John, and Nyberg, Karl, "Ada Verification Using Existing Tools", Proceedings of Verkshop III, to appear in *Software Engineering Notes.*

[Melliar-Smith82] - Melliar-Smith, P.M. and Schwartz, Richard, *Formal Specification and Mechanical Verification of SIFT: A Fault-Tolerant Flight Control System,* Technical Report CSL-133, SRI International, January 1982.

[Melliar-Smith85] - Melliar-Smith, Michael, and Rushby, John *The Enhanced HDM System for Specification and Verification,* Private Communication, Computer Science Laboratory, SRI International, 333 Ravenswood, Menlo Park, CA, June, 1985.

[Millen82] - Millen, Jonathan K. and Drake, David L., "An Experiment with Affirm and HDM", *The Journal of Systems and Software 2, 159-175,* 1981.

[Moriconi77] - Moriconi, Mark S., *A System of Incrementally Designing & Verifying Programs,* ICSCA-CMP-9, Institute for Computing Science, The University of Texas at Austin, Austin, TX 78712, December, 1977.

[Musser] - Musser, David R., *Aids to Hierarchical Specification Structuring and Reusing Theorems in AFFIRM-85,* Proceedings of the Verkshop II, February, 1985.

[Odyssey84] - Oddysey Research Associates, Inc. *A Verifiable Subset of Ada,* (Revised Preliminary Report) Oddysey Research Associates, Inc., 713 Clifton St., Ithaca, NY 14850, 14 September 1984.

[PDL] - *SURVEY OF Ada-BASED PDLS (FINAL REPORT),* Computer Technology Associates, Inc., 7927 Jones Branch Drive, Suite 600W, McLean, VA 22102, January 1985.

[Pneuli82] - Fneuli, A, and deRoever, W. P., "Rendesvous with Ada — A Proof Theoretical View", *Proceedings of the AdaTEC Conference on Ada,* Arlington, Va., pp 129-137, October 1982.

[Reps84] - REps, Thomas and Alpern, Bowen, "Interactive Proof Checking", in *Proceedings of the 11th ACM Symposium on POPL,* pp. 36-45, Salt Lake City, Utah, January 15-18, 1984.

[Roubine76] - Roubine, Olivier, *The Design and Use of Specification Languages,* Technical Report CSL-48, Stanford Research Institute, Menlo Park, CA 94025, October 1976.

[Rowe] - Rowe, Kenneth E., *The U. S. DoD Computer Security Center and Ada*, (unpublished manuscript), Office of Research and Development, DoD Computer Security Center, Fort Meade, MD 20755-6000, April 1985.

[SIFT] - *Proceedings of a Formal Verification/Design Proof Peer Review*, RTI/2094/13-01F, Validation Methods Research for Fault-Tolerant Avionics and Control System Sub-Working Groups Meeting, Fault Tolerant Computing Program, Center for Digital Systems Research, Research Triangle Institute, Research Triangle Park, NC 27709, January, 1984.

[Smith84] - Smith, Lawrence M. and Siebert, Ann E., *KAIS FEU Requirements*, Institute for Computing Science, University of Texas at Austin, Austin, TX 78712, October 1984.

[Smith81] - Smith, Michael K., Siebert, Ann, DiVito, Bendetto. and Good, Donald I., "A Verified Encrypted Packet Interface", *Software Engineering Notes*, Volume 6, Number 3, July 1981.

[Smith83] - Smith, Michael K., *Model and Design Proofs in Gypsy: An Example Using Bell and LaPadula*, Institute for Computing Science, The University of Texas at Austin, Austin, TX 78712, February 1983.

[TCSEC83] - *Department of Defense Trusted Computer System Evaluation Criteria*, CSC-STD-001-83, Department of Defense Computer Security Center, Ft. George G. Meade, MD 20755, 15 August 1983.

[Tripathi80] - Tripathi, Anand R., Young, William D., Good, Donald I., "A Preliminary Evaluation of Verifiability in Ada", *Proceedings of the ACM National Conference*, Nashville, TN, October 1980.

[vonNeumann61] - John von Neumann, "Planning and Coding Problems for an Electronic Computing Instrument" in Taub, A. H. (editor), *John von Neumann, Collected Works, Volume V*, pp. 80-235, Pergamon, 1961.

[Young80] - Young, William D., Good, Donald I., "Generics and Verification in Ada", *Proceedings of the ACM Symposium on the Ada Language*, Boston, MA, pp. 123-127, 9-11 December 1980.

[Young81] - Young, William D., Good, Donald I., "Steelman and the Verifiability of (Preliminary) Ada", *ACM SIGPLAN Notices*, 16(2):113-119, February 1981.

Distribution List for P-1859

Ms. Virginia Castor
Director, Ada Joint Program Office
1211 Fern Street, Room C-107
Arlington, VA 22202

Mr. George Cowan
Verdix Corporation
14130A Sullylfield Circle
Chantilly, VA 22021

Mr. Don Milton
Verdix Corporation
14130A Sullylfield Circle
Chantilly, VA 22021

Mr. Karl Nyberg
Verdix Corporation
14130A Sullylfield Circle
Chantilly, VA 22021

Defense Technical Info. Center (2 copies)
Cameron Station
Alexandria, VA 22314

DoD-IDA Management Office
1801 N. Beauregard St.
Alexandria, VA 22311

CSED Review Panel

Dr. Dan Alpert, Director
Center for Advanced Study
University of Illinois
912 W. Illinois Street
Urbana, Illinois 61801

Dr. Barry W. Boehm
TRW Defense Systems Group
MS 2-2304
One Space Park
Redondo Beach, CA 90278

Dr. Ruth Davis
The Pymatuning Group, Inc.
2000 N. 15th Street, Suite 707
Arlington, VA 22201

Dr. Larry E. Druffel
Rational Machines
1501 Salado Drive
Mountain View, CA 94043

Dr. C.E. Hutchinson, Dean
Thayer School of Engineering
Dartmouth College
Hanover, NH 03755

Mr. Oliver Selfridge
45 Percy Road
Lexington, MA 02173

IDA

Gen. W.Y. Smith, HQ
Mr. Seymour Deitchman, HQ
Mr. Robin Pirie, HQ
Ms. Karen Weber, HQ
Dr. Jack Kramer, CSED
Dr. John Salasin, CSED
Dr. Robert Winner, CSED
Ms. Audrey A. Hook, CSED (2 copies)
Mr. Terry Mayfield, CSED
Mr. Max Robinson, CSED
Mr. Clyde Roby, CSED
Ms. Katydean Price, CSED (2 copies)
IDA Control & Dist. Vault (15 copies)

END

FILMED

5-86

DTIC